Simulating the DPLL(T) procedure in a sequent calculus with focusing

Mahfuza Farooque, Stéphane Lengrand, Assia Mahboubi

Simulating the DPLL($\mathcal{T}$) procedure in a sequent calculus with focusing

Mahfuza Farooque[1,3], Stéphane Lengrand[1,3], and Assia Mahboubi[2,3]

[1]CNRS, France
[2]INRIA, France
[3]École Polytechnique, France

Abstract

This paper gives an abstract description of decision procedures for Satisfiability Modulo Theory (SMT) as proof search procedures in a sequent calculus with polarities and focusing. In particular, we show how to simulate the execution of standard techniques based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure modulo theory as the gradual construction of a proof tree in sequent calculus.

The construction mimicking a run of DPLL-modulo-Theory can be obtained by a meta-logical control on the proof-search in sequent calculus. This control is provided by polarities and focusing features, which therefore narrow the corresponding search space in a sense we discuss. This simulation can also account for backjumping and learning steps, which correspond to the use of general cuts in sequent calculus.

1 Introduction

Satisfiability Modulo Theories (SMT) is a family of problems that generalises SAT-problems: instead of considering the satisfiability of conjunctive normal forms (CNF) over propositional variables, SMT problems are concerned with the satisfiability of CNF over atomic propositions from a theory such as linear arithmetic or bit vectors.

Given a procedure deciding the consistency -with respect to such a theory- of a conjunction of atoms or negated atoms, SMT-solving organises a cooperation between this procedure and SAT-solving techniques, thus providing a decision procedure for SMT-problems.

This smart extension of the successful SAT-solving techniques opened a prolific area of research and led to the implementation of ever-improving tools, called SMT-solvers, now crucial to a number of applications in software verification. The architecture of SMT-solvers is based on the extension of the Davis, Putnam, Logemann and Loveland (DPLL) procedure [DP60, DLL62] for solving SAT-problems to a procedure called DPLL($\mathcal{T}$) [NOT06] addressing SMT-problems.

This paper describes how to transpose DPLL($\mathcal{T}$) into a proof-theoretical framework, namely a sequent calculus, where the SAT-solving techniques within

$\mathsf{DPLL}(\mathcal{T})$ are transposed as the application of a standard proof-search process, i.e. the incremental construction of a proof-tree.

The motivation for doing this is threefold:

- Firstly, provide an abstract understanding of $\mathsf{DPLL}(\mathcal{T})$ implementations in proof-theoretical terms (offering e.g. a new angle to the questions of soundness and completeness of the procedure).
- Secondly, offer a new starting point for the incorporation of SMT-solving into proof assistant software based on proof theory. We aim here at relying on the direct application of the standard proof-search mechanisms (pertaining to the logic on which the software is based), rather than relying on specific implementations of $\mathsf{DPLL}(\mathcal{T})$: For instance with the proof-assistant Coq [Coq], recent literature provides an internal but specific implementation using boolean reflection [LC09], as well as a technique that calls an external SMT-solver as a blackbox and re-interprets its trace [AFG$^+$11].
- Thirdly, offer new leads on how to generalise SMT-techniques to a wider class of problems, or combine them with other techniques from automated reasoning: the proof-theoretical framework to which we transpose $\mathsf{DPLL}(\mathcal{T})$ supports much more expressive logics than the class of SMT-problems (e.g. with quantifiers) and, as we aim at transposing into it other techniques (e.g. strong and weak connection tableaux, superposition calculi, etc), we hope to turn this framework into a useful platform for combining them.

To be clear, we do not claim, until the third point above is successfully achieved, that the present paper improves in any way SMT-solving techniques or implementations. However we do aim at formalising this bridge between different computer science areas.

The description of $\mathsf{DPLL}(\mathcal{T})$ in an abstract framework has been studied by [NOT05] in order to formally reason about the non-trivial properties of the algorithms implemented by the SMT tools. For that purpose, a transition system based on rewrite rules emerged as the most appropriate approach [NOT05, NOT06]. Attempts based on sequent calculus showed that important methodological features of $\mathsf{DPLL}(\mathcal{T})$ like backjumping or lemma learning were more difficult to capture in the somehow more rigid setting of root-first decomposition of formulae and derivation trees.

However, the proof-theoretic framework that we use in this paper to simulate $\mathsf{DPLL}(\mathcal{T})$ *is* a sequent calculus. Sequent calculus represents proofs using inference steps that decompose the connectives present in the goal to be proved. While Gentzen's original rules offer a lot of non-determinism in the proof-search-space, a tighter control on proof-search is provided by more recent features: *polarities* and *focusing*. These arose from Linear Logic [Gir87, And92], but also make sense in classical logic [Gir91, Lau03, LM09]. In brief, the inference rules decomposing the connectives of the same polarity can be chained without loosing completeness - this considerably narrows the search space (see e.g. [MNPS91, And92]).

In this paper we show how a sequent calculus with polarities, focusing and cut rules allows for a precise simulation of the basic and advanced versions of $\mathsf{DPLL}(\mathcal{T})$ described in [NOT06], including some features that previous work (e.g. [Tin02]) eluded or left implicit. This sequent calculus is a variant of the

system LKF of [LM09] for propositional polarised classical logic. $\mathsf{LK}^p(\mathcal{T})$ differs from LKF in three ways:

- System LKF assumes that all atoms come with a pre-determined *polarity*: positive or negative. $\mathsf{LK}^p(\mathcal{T})$ allows the polarisation of atoms *on-the-fly*: the root of a proof-tree might have none of its atoms polarised, but atoms may become positive or negative in sub-trees, closer to the leaves.

- We allow an *analytic cut-rule*, despite the fact that proof-search in sequent calculus is usually done in a cut-free system, as the cut-rule usually unreasonably widens the search-space. Analytic cuts only concern atomic cut-formulae among the finitely many atoms present in the rest of the sequent. Allowing them does widen the search space (which might be narrowed in other ways) but this sometimes permits to draw quicker conclusions, in a way similar to $\mathsf{DPLL}(\mathcal{T})$'s Decide rule. Miller and Nigam [MN07] have already shown how to use analytic cuts to incorporate *tables* into proofs, i.e. make sure that, once an atom is proved or known to be true (from a table of lemmas), the subsequent proof-search never tries to re-prove it. This is achieved by giving, to the atom that is cut, two opposite polarities in the two premisses of the cut. The simulation of $\mathsf{DPLL}(\mathcal{T})$'s Decide rule in $\mathsf{LK}^p(\mathcal{T})$ will use the same trick.

- Finally, $\mathsf{LK}^p(\mathcal{T})$ represents proofs *modulo a theory*, which means that some of the syntactic checks that are traditionally needed to show some inference steps as valid instances of inference rules, are replaced by semantical conditions possibly checked by an external procedure (i.e. the same as for $\mathsf{DPLL}(\mathcal{T})$).

  For instance, when proving that $\Gamma \vdash p(1+1)$ is an axiom, instead of requring $p(1+1)$ to be in Γ, we also accept to find only $p(2)$ in Γ. This of course requires a theory where $1+1$ and 2 are identified.

Non-deterministic features of $\mathsf{DPLL}(\mathcal{T})$ like decision/backtracking or learning/backjumping are related to different flavours of cut rules. The control of the proof-search space in this sequent presentation is ensured by the polarisation features of the calculus which capture the efficiency of the $\mathsf{DPLL}(\mathcal{T})$ procedure.

The paper is organised as follows: Section 2 introduces the focused sequent calculus with polarities, Section 3 reviews the basic $\mathsf{DPLL}(\mathcal{T})$ procedure [NOT05, BNOT06], Section 4 presents the simulation of $\mathsf{DPLL}(\mathcal{T})$ in sequent calculus and Section 5 presents the extension of $\mathsf{DPLL}(\mathcal{T})$ with Backjump and Lemma learning rules and its simulation.

2 The $\mathsf{LK}^p(\mathcal{T})$ sequent calculus with analytic cuts

In this section we introduce a focused sequent calculus called $\mathsf{LK}^p(\mathcal{T})$, designed for propositional polarised classical logic *modulo a theory*, as described in the introduction.

This sequent calculus (and this logic) involve a notion of *literals* and a notion of *theory*. The reader can safely see behind this terminology the standard notions from proof theory and automated reasoning; we will use them later on. However at this point, very little is required from or assumed about those two notions:

Definition 2.1 (Literals) *Let $\mathcal{L}$ be a set of elements called* literals*, equipped with an involutive function called* negation *from $\mathcal{L}$ to $\mathcal{L}$. In the rest of this paper, a possibly primed or indexed lowercase l always denotes a literal, and $l^\perp$ its negation.*

The second ingredient of $\mathsf{LK}^p(\mathcal{T})$ is a theory, or more precisely a notion of *(in)consistency modulo a theory*:

Definition 2.2 (Inconsistency predicate, syntactical inconsistency) *An* inconsistency predicate *is a predicate over sets of literals*

- *that is upward closed (if a subset of a set satisfies the predicate, so does the set)*
- *satisfied by the set $\{l, l^\perp\}$ for every literal l.*

The smallest inconsistency predicate is called the syntactical inconsistency *predicate. If a set $\mathcal{P}$ of literals satisfies the syntactically inconsistency predicate, we say that $\mathcal{P}$ is* syntactically inconsistent*, denoted $\mathcal{P} \models$. Otherwise $\mathcal{P}$ is* syntactically consistent*.*

Besides syntactical inconsistency, we now consider an(other) inconsistency predicate called *semantical inconsistency* or *inconsistency modulo theory*. In this abstract setting, this is the description of the backend theory we require and rely on.

Definition 2.3 (Semantical inconsistency) *If a set $\mathcal{P}$ of literals satisfies the semantical inconsistency predicate, we say that $\mathcal{P}$ is* semantically inconsistent *or* inconsistant modulo theory*, denoted by $\mathcal{P} \models_{\mathcal{T}}$. Otherwise $\mathcal{P}$ is* semantically consistent *or* consistant modulo theory*.*

Definition 2.4 (Formulae, negation) *The formulae of propositional polarised classical logic are given by the following grammar:*

$$\textit{Formulae} \quad A, B, \ldots \quad ::= \quad l \mid A \wedge^+ B \mid A \vee^+ B \mid A \wedge^- B \mid A \vee^- B$$

where l ranges over literals.

The size of a formula A, denoted $\sharp(A)$, is its size as a tree (number of nodes).

Let $\mathcal{P} \subseteq \mathcal{L}$ be syntactically consistent.

We define $\mathcal{P}$-positive formulae and $\mathcal{P}$-negative formulae as the formulae generated by the following grammars:

$$\begin{array}{llll} \mathcal{P}\textit{-positive formulae} & P, \ldots & ::= & p \mid A \wedge^+ B \mid A \vee^+ B \\ \mathcal{P}\textit{-negative formulae} & N, \ldots & ::= & p^\perp \mid A \wedge^- B \mid A \vee^- B \end{array}$$

where p ranges over $\mathcal{P}$.

Negation is recursively extended into a involutive map from formulae to formulae as follows:

$(A \wedge^+ B)^\perp$	$:= A^\perp \vee^- B^\perp$	$(A \wedge^- B)^\perp$	$:= A^\perp \vee^+ B^\perp$
$(A \vee^+ B)^\perp$	$:= A^\perp \wedge^- B^\perp$	$(A \vee^- B)^\perp$	$:= A^\perp \wedge^+ B^\perp$

Remark 2.1 *Note that, given a syntactically consistent set $\mathcal{P}$ of literals, negations of $\mathcal{P}$-positive formulae are $\mathcal{P}$-negative and vice versa.*

In the rest of this paper, a possibly primed or indexed Γ always denotes a set of formulae. By $\mathsf{lit\_ctxt}(\Gamma)$ we denote the subset of elements of Γ that are literals.

Definition 2.5 (System $\mathsf{LK}^p(\mathcal{T})$) *The sequent calculus $\mathsf{LK}^p(\mathcal{T})$ has two kinds of sequents:*

$$\Gamma \vdash [P] \qquad \textit{where } \mathrm{P} \textit{ is in the } \text{focus } \textit{of the sequent}$$
$$\Gamma \vdash \Gamma'$$

Its rules, given in Figure 1, fall in three categories: synchronous *rules,* asynchronous *rules and* structural *rules.*

$$\frac{\Gamma \vdash^{\mathcal{P}} [A] \qquad \Gamma \vdash^{\mathcal{P}} [B]}{\Gamma \vdash^{\mathcal{P}} [A \wedge^{+} B]} \qquad \frac{\Gamma \vdash^{\mathcal{P}} [A_i]}{\Gamma \vdash^{\mathcal{P}} [A_1 \vee^{+} A_2]}$$

$$\frac{}{\Gamma \vdash^{\mathcal{P},p} [p]}\ \mathsf{lit\_ctxt}(\Gamma), p^{\perp} \models_{\mathcal{T}} \qquad \frac{\Gamma \vdash^{\mathcal{P}} N}{\Gamma \vdash^{\mathcal{P}} [N]}\ N \text{ is } \mathcal{P}\text{-negative}$$

$$\frac{\Gamma \vdash^{\mathcal{P}} A, \Gamma' \qquad \Gamma \vdash^{\mathcal{P}} B, \Gamma'}{\Gamma \vdash^{\mathcal{P}} A \wedge^{-} B, \Gamma'} \qquad \frac{\Gamma \vdash^{\mathcal{P}} A_1, A_2, \Gamma'}{\Gamma \vdash^{\mathcal{P}} A_1 \vee^{-} A_2, \Gamma'} \qquad \frac{\Gamma, A^{\perp} \vdash^{\mathcal{P}} \Gamma'}{\Gamma \vdash^{\mathcal{P}} A, \Gamma'}\ A \text{ is } \mathcal{P}\text{-positive or literal}$$

$$\frac{\Gamma \vdash^{\mathcal{P},l}}{\Gamma \vdash^{\mathcal{P}}} \qquad \frac{\Gamma, P^{\perp} \vdash^{\mathcal{P}} [P]}{\Gamma, P^{\perp} \vdash^{\mathcal{P}}}\ P \text{ is } \mathcal{P}\text{-positive} \qquad \frac{}{\Gamma \vdash^{\mathcal{P}}}\ \mathsf{lit\_ctxt}(\Gamma) \models_{\mathcal{T}}$$

Figure 1: System $\mathsf{LK}^p(\mathcal{T})$

Asynchronous rules are invertible and can always be applied eagerly while trying to construct the proof tree of a given sequent. When no more negative rule applies, a clever choice must be made to put a positive formula in focus before applying the corresponding synchronous rule. Each such rule has the focused formula in the positive sequent as the principal formula, and if the operands of the principal connective are also positive then the focus is maintained on them in their corresponding premises.

Besides these standard features, two rules of the $\mathsf{LK}^p(\mathcal{T})$ calculus can call for a decision procedure associated with the background theory $\mathcal{T}$, under the form of its semantical inconsistency check $\models_{\mathcal{T}}$. First, we can use this decision procedure to close a branch in which the context Γ is semantically inconsistant. Second, we can also close the current branch if a positive literal currently under focus is semantically inconsistent with the literals $\mathsf{lit\_ctxt}(\Gamma)$ of the current context.

As discussed in the introduction, we also consider the following *analytic cut-rule*:

$$\frac{\Gamma, l \vdash^{\mathcal{P}} \qquad \Gamma, l^{\perp} \vdash^{\mathcal{P}}}{\Gamma \vdash^{\mathcal{P}}}$$

with the condition that l or $l^{\perp}$ appears as a subformula of Γ.

Definition 2.6 (Size of proof-trees in $\mathsf{LK}^p(\mathcal{T})$) *The size of a proof-tree in $\mathsf{LK}^p(\mathcal{T})$ is its number of nodes, i.e. the number of rule applications its construction necessitates.*

3 The basic $\mathsf{DPLL}(\mathcal{T})$ algorithm

In this section we review the basic $\mathsf{DPLL}(\mathcal{T})$ algorithm.

Intuitively, $\mathsf{DPLL}(\mathcal{T})$ aims at proving the inconsistency of a set of *clauses* with respect to a theory. We therefore retain from the previous section the notions of literals and the notions of inconsistencies, and introduce clauses:

Definition 3.1 (Clause) *A* clause *is a finite disjunction of literals, considered up to permutation.*

In the rest of the paper, a possibly indexed upper cased C always denotes a clause. The empty clause is denoted by $\bot$. The number of literals in a clause C is denoted $\sharp(C)$. The possibly indexed symbol ϕ always denotes finite sets of clauses $\{C_1, \ldots, C_n\}$, which can also be seen as a Conjunctive Normal Form (CNF). We use $\sharp(\phi)$ to denote the sum of the sizes of the clauses in ϕ. Finally $\mathsf{lit}(\phi)$ denotes the set of literals that appear in ϕ.

Definition 3.2 (Decision literals and sequences)

We consider an isomorphic copy $\mathcal{L}^d$ of the set $\mathcal{L}$ of literals, whose elements are called decision literals*, i.e. a tagged version of the literals in $\mathcal{L}$. Decision literals are denoted by l^d.*

We use the possibly indexed symbol Δ to denote a finite sequence of possiblity tagged literals, with $\emptyset$ denoting the empty sequence. We also use Δ_1, Δ_2 and Δ_1, l, Δ_2 to denote the suggested concatenation of sequences.

For such a sequence Δ, we write $|\Delta|$ for the subset of $\mathcal{L}$ containing all the literals in Δ with their potential tags removed. By construction, the sequences we consider will always be duplicate-free, so the difference between Δ and $|\Delta|$ is just a matter of tags and ordering.

Definition 3.3 (Syntactic entailment) *Let Δ be a set of literals (or a sequence seen as its corresponding set $|\Delta|$) and C a clause. We say that Δ* syntactically entails *$\neg C$, denoted $\Delta \models \neg C$, if for all $l \in \mathsf{lit}(C)$ we have $l^\bot \in \Delta$.*

Definition 3.4 (Semantic entailment) *For every set Δ of literals (or a sequence seen as its corresponding set $|\Delta|$), the* set of untagged literals that are semantically entailed *by Δ is $\mathsf{Sat}(\Delta) := \{l \mid \Delta, l^\bot \models_{\mathcal{T}}\}$. For any set of clauses ϕ, the set of literals occuring in ϕ that are semantically entailed by Δ is denoted by $\mathsf{Sat}_\phi(\Delta) := \mathsf{Sat}(\Delta) \cap \mathsf{lit}(\phi)$.*

Remark 3.1 *Semantical consequences are the analogous of the consequences of a partial boolean assignment in the context of a DPPL procedure for propositional logic without theory. Obviously, if $l \in \Delta$, then $l \in \mathsf{Sat}(\Delta)$. If $\phi_1 \subseteq \phi_2$, then for any Δ, $\mathsf{Sat}_{\phi_1}(\Delta) \subseteq \mathsf{Sat}_{\phi_2}(\Delta)$.*

We are now ready to describe the basic $\mathsf{DPLL}(\mathcal{T})$ procedure as a transition binary relation between states. We purposely follow closely the presentation given in [NOT06].

Definition 3.5 (Basic $\mathsf{DPLL}(\mathcal{T})$) *A* state *of the $\mathsf{DPLL}(\mathcal{T})$ procedure is either the state UNSAT, or a pair denoted $\Delta \| \phi$, where ϕ is a set of clauses and Δ is a sequence of possibly tagged literals. The* transition rules *of the $\mathsf{DPLL}(\mathcal{T})$ procedure are:*

- *Fail:*
  $\Delta \| \phi, C \Rightarrow \mathsf{UNSAT}$, *with $\Delta \models \neg C$ and there is no decision literal in Δ.*

- *Decide:*
  $\Delta \| \phi \Rightarrow \Delta, l^d \| \phi$ *where* $l \notin \Delta$, $l^\perp \notin \Delta$, $l \in \mathit{lit}(\phi)$.

- *Backtrack:*
  $\Delta_1, l^d, \Delta_2 \| \phi, C \Rightarrow \Delta_1, l^\perp \| \phi, C$ *if* $\Delta_1, l, \Delta_2 \models \neg C$ *and no decision literal is in* Δ_2.

- *Unit propagation:*
  $\Delta \| \phi, C \vee l \Rightarrow \Delta, l \| \phi, C \vee l$ *where* $\Delta \models \neg C$, $l \notin \Delta$, $l^\perp \notin \Delta$.

- *Theory Propagate:*
  $\Delta \| \phi \Rightarrow \Delta, l \| \phi$ *where* $l \in \mathsf{Sat}_\phi(\Delta)$ *and* $l \notin \Delta, l^\perp \notin \Delta$.

4 Simulation of the basic DPLL($\mathcal{T}$) algorithm in sequent calculus

The aim of this section is to describe how the basic DPLL($\mathcal{T}$) procedure can be transposed into a proof-search process for sequents of the LK^p($\mathcal{T}$) calculus. A complete and successful run of the DPLL($\mathcal{T}$) procedure is a sequence of transitions $\emptyset \| \phi \Rightarrow^* \mathsf{UNSAT}$, which ensures that the set of clauses ϕ is inconsistent modulo theory. Hence, we are devising a proof-search process aiming at building an LK^p($\mathcal{T}$) proof-tree for sequents of the form $\phi' \vdash$, where ϕ' represents the set of clauses ϕ as a sequent calculus structure, in the following sense:

Definition 4.1 (Representation of clauses as formulae) *An LK^p($\mathcal{T}$) formula C'* represents *a DPLL($\mathcal{T}$) clause C if $C' = l_1 \vee^- \ldots \vee^- l_p$ with $\{l_j\}_{j=1\ldots p} = \mathit{lit}(C)$.*

A set of formulae ϕ' represents a set of clauses ϕ if the formulae in ϕ' pairwise represent the clauses in ϕ.

Remark 4.1 *If C' represents C, then $\sharp(C') \leq 2\sharp(C)$ (there are fewer symbols $\vee^-$ than there are literals in $\mathit{lit}(C)$).*

Note here that we carefully use the negative disjunction connective to translate DPLL($\mathcal{T}$) clauses. This plays a central role in controlling the proof-search process, not only to mimic DPLL($\mathcal{T}$) without duplicating formulae but more generally to control the search space.

Now, in order to construct a proof of $\phi' \vdash$ from a run $\emptyset \| \phi \Rightarrow^* \mathsf{UNSAT}$, we proceed incrementally by considering the intermediate steps of the DPLL($\mathcal{T}$) run:

$$\emptyset \| \phi \Rightarrow^* \Delta \| \phi \Rightarrow^* \mathsf{UNSAT}$$

In the intermediate DPLL($\mathcal{T}$) state $\Delta \| \phi$, the sequence Δ is a log of both the search space explored so far (in $\emptyset \| \phi \Rightarrow^* \Delta \| \phi$) and the search space that remains to be explored (in $\Delta \| \phi \Rightarrow^* \mathsf{UNSAT}$). In this log, a tagged decision literal l^d indicates a point where the procedure has made an exploratory choice (the case where l is true has been/is being explored, the case where $l^\perp$ is true remains to be explored), while untagged literals in Δ are predictable consequences of the decisions made so far and of the set of clauses ϕ to be falsified.

If we are to express the DPLL($\mathcal{T}$) precedure as the incremental construction of a LK^p($\mathcal{T}$) proof-tree, we should get from $\emptyset \| \phi \Rightarrow^* \Delta \| \phi$ a proof-tree that is not yet complete and get from $\Delta \| \phi \Rightarrow^* \mathsf{UNSAT}$ some (complete) proof-tree(s) that

can be "plugged into the holes" of the incomplete tree. We should read in Δ the "interface" between the incomplete tree that has been constructed and the complete sub-trees to be constructed.

We use the plural here since there can be more than one sub-tree left to construct: $\Delta \| \phi \Rightarrow^* \mathsf{UNSAT}$ contains the information to build not only a proof of $|\Delta|, \phi' \vdash$, but also proofs of the sequents corresponding to the other parts of the search space to be explored, characterised by the tagged literals in Δ.

For instance, a run from $1, 2^d, 3, 4^d \| \phi \Rightarrow^* \mathsf{UNSAT}$ contains the information to build a proof of $1, 2, 3, 4, \phi' \vdash$ but also the proofs of $1, 2, 3, 4^\perp, \phi' \vdash$ and $1, 2^\perp, \phi' \vdash$.

Those extra sequents are obtained by collecting from a sequence Δ its "backtrack points" as follows:

Definition 4.2 (Backtrack points) *The* backtrack points $[\![\Delta]\!]$ *of a sequence* Δ *of possibly tagged literals is the set of sets of untagged literals recursively defined by the rules of Fig 2, where again* $|\Delta|$ *denotes the set of untagged literals obtained by erasing the tags in the sequence* Δ*.*

$$
\begin{array}{lcl}
[\![()]\!] & := & \emptyset \\
[\![\Delta, l]\!] & := & [\![\Delta]\!] \\
[\![\Delta, l^d]\!] & := & [\![\Delta, l^\perp]\!] \cup \{|\Delta, l^\perp|\}
\end{array}
$$

Figure 2: Collecting backtrack points

Remark 4.2 *If* Δ *features* n *decision literals, then* $[\![\Delta]\!]$ *contains* n *elements. Each element of* $[\![\Delta]\!]$ *can in fact be associated with a decision literal* l^d *in* $\Delta = \Delta_1, l^d, \Delta_2$ *and consists of the prefix of* Δ *which subsists after a backtrack transition.*

Now, coming back to the $\mathsf{DPLL}(\mathcal{T})$ transition sequence $\emptyset \| \phi \Rightarrow^* \Delta \| \phi$ and its intuitive counterpart in sequent calculus, we have to formalise the notion of incomplete, or *partial*, proof-tree together with the notion of "filling its holes":

Definition 4.3 (Partial proof-tree, extension of a partial proof-tree) *A* partial proof-tree *in* $\mathsf{LK}^p(\mathcal{T})$ *is a tree labelled with sequents without focus or right-hand side, whose leaves are tagged as either* open *or* closed*, and such that every node that is not an open leaf, together with its children, form an instance of the* $\mathsf{LK}^p(\mathcal{T})$ *rules. The* size *of a partial proof-tree is its number of nodes.*

A partial proof-tree π' *is an* n-extension *of* π *if* π' *is obtained from* π *by replacing one of its open leaves labelled with a sequent* s *by a partial proof-tree of size at most* n *and whose conclusion is* s*.*

Remark 4.3 *A partial proof-tree that has no open leaf is (isomorphic to) a well-formed complete* $\mathsf{LK}^p(\mathcal{T})$ *proof of the sequent labelling its root. In that case, we say the proof-tree is* complete*.*

The intuition that an intermediate $\mathsf{DPLL}(\mathcal{T})$ state describes an "interface" between a partial proof-tree and the complete proof-trees that should be plugged into its holes, is formalised as follows:

Definition 4.4 (Correspondence between $\mathsf{DPLL}(\mathcal{T})$ states and $\mathsf{LK}^p(\mathcal{T})$ partial proof-trees) *A partial proof-tree* π corresponds to *a* $\mathsf{DPLL}(\mathcal{T})$ *state* $\Delta \| \phi$ *if:*

- *there is a one-to-one correspondence between the open leaves of π and the elements of $[\![\Delta]\!] \cup \{|\Delta|\}$;*
- *the sequent label of the open leaf corresponding to a set $\Delta_0 \in [\![\Delta]\!] \cup \{|\Delta|\}$ is of the form $\Delta', \phi' \vdash^{\Delta_0}$, where:*
  - *ϕ' represents ϕ in the sense of definition 4.1;*
  - *$\mathsf{Sat}_\phi(\Delta_0) = \mathsf{Sat}_\phi(\Delta')$.*

A partial proof-tree π corresponds to *the state UNSAT if it has no open leaf.*

Remark 4.4 *In the general case, different partial proof-trees might correspond to a same $\mathsf{DPLL}(\mathcal{T})$ state (just like different $\mathsf{DPLL}(\mathcal{T})$ runs may reach that state from the initial one).*

Note that we do not require anything from the conclusion of a partial proof-tree corresponding to $\Delta\|\phi$: just as our correspondence says nothing about the $\mathsf{DPLL}(\mathcal{T})$ transitions taking place after $\Delta\|\phi$ (nor about the trees to be plugged into the open leaves), it says nothing about the transitions taking place before $\Delta\|\phi$ (nor about the incomplete proof-tree, except for its open leaves).

If a partial proof-tree π corresponds to a $\mathsf{DPLL}(\mathcal{T})$ state $\Delta\|\phi$ where there is no decision literals in Δ, then there is exactly one open leaf in π, and it is labelled by a sequent of the form $\Delta', \phi' \vdash^{|\Delta|}$, where ϕ' represents ϕ and $\mathsf{Sat}_\phi(\Delta) = \mathsf{Sat}_\phi(\Delta')$.

To the initial state $\emptyset\|\phi$ of a run of the $\mathsf{DPLL}(\mathcal{T})$ procedure corresponds the partial proof-tree consisting of one node (both root and open leaf) labelled with the sequent $\phi' \vdash$, where ϕ' represents ϕ.

The simulation theorem below provides a systematic way to interpret any $\mathsf{DPLL}(\mathcal{T})$ transition as a transformation on partial-proof trees which preserves the correspondences given in Definition 4.4 and controls the growth of the proof trees.

Theorem 4.1 (Simulation of $\mathsf{DPLL}(\mathcal{T})$ in $\mathsf{LK}^p(\mathcal{T})$) *Let $\Delta\|\phi \Rightarrow \mathcal{S}_2$ be a valid $\mathsf{DPLL}(\mathcal{T})$ transition, and π_1 be a partial proof tree in $\mathsf{LK}^p(\mathcal{T})$ corresponding to $\Delta\|\phi$.*

There is, in $\mathsf{LK}^p(\mathcal{T})$, a partial proof tree π_2 corresponding to $\mathcal{S}_2$ such that π_2 is a $(2\sharp(\phi) + 3)$-extension of π_1.

Proof. By case analysis on the nature of the transition:

- Fail: $\Delta\|\phi, C \Rightarrow \mathsf{UNSAT}$ with $\Delta \models \neg C$ and there is no decision literal in Δ.

  Let π_1 be a partial proof-tree corresponding to $\Delta\|\phi, C$. Since there are no decision literals in Δ, π_1 has exactly one open leaf, and it is labelled by $\Delta', \phi', C' \vdash^{|\Delta|}$ where ϕ' represents ϕ, C' represents C and $\mathsf{Sat}_{\phi,C}(\Delta) = \mathsf{Sat}_{\phi,C}(\Delta')$. Let $C = l_1 \vee \ldots \vee l_n$.

  From $\Delta \models \neg C$ we get $\forall i,\ l_i^\perp \in |\Delta| \subseteq \mathsf{Sat}_{\phi,C}(\Delta) = \mathsf{Sat}_{\phi,C}(\Delta')$.

  We extend π_1 into π_2 by replacing the open leaf by the following (complete) proof-tree:

$$\frac{\left(\begin{array}{c}\overline{\Delta', \phi', C' \vdash^{|\Delta|} [l_i^{\perp}]}\end{array}\right)_{l_i \in \mathsf{lit}(C)} \quad \vdots \wedge^{+}. \quad \Delta', \phi', C' \vdash^{|\Delta|} [C'^{\perp}]}{\Delta', \phi', C' \vdash^{|\Delta|}}$$

The top rules can be applied since $l_i^{\perp} \in |\Delta|$ and $\mathsf{lit\_ctxt}(\Delta', \phi', C'), l_i \models_{\mathcal{T}}$. All the leaves are closed. π_2 is a $\sharp(C') + 1$-extension of π_1 that is complete and therefore corresponds to the UNSAT state of the $\mathsf{DPLL}(\mathcal{T})$ run (and note that $\sharp(C') \leq 2\sharp(\phi)$).

- Decide: $\Delta \| \phi \Rightarrow \Delta, l^d \| \phi$ where $l \notin \Delta$, $l^{\perp} \notin \Delta$, $l \in \mathsf{lit}(\phi)$.

  Let π_1 be a partial proof-tree corresponding to $\Delta \| \phi$. The open leaf corresponding to $|\Delta|$ is of the form $\Delta', \phi' \vdash^{|\Delta|}$ where ϕ' represents ϕ and $\mathsf{Sat}_{\phi}(|\Delta|) = \mathsf{Sat}_{\phi}(\Delta')$.

  We extend π_1 into π_2 by replacing this leaf by the following (partial) proof-tree:

  $$\frac{\dfrac{\Delta', l^{\perp}, \phi' \vdash^{|\Delta|, l^{\perp}}}{\Delta', l^{\perp}, \phi' \vdash^{|\Delta|}} \quad \dfrac{\Delta', l^{\perp}, \phi' \vdash^{|\Delta|, l}}{\Delta', l, \phi' \vdash^{|\Delta|}}}{\Delta', \phi' \vdash^{|\Delta|}}$$

  Note that we use here the analytic cut rule of $\mathsf{LK}^p(\mathcal{T})$. π_2 is a 3-extension of π_1 that corresponds to $\Delta, l^d \| \phi$. Indeed, we have $[\![\Delta, l^d]\!] \cup \{|\Delta, l^d|\} = [\![\Delta]\!] \cup \{(|\Delta|, l^{\perp})\} \cup \{(|\Delta|, l)\}$ and $\mathsf{Sat}_{\phi}(|\Delta|, l) = \mathsf{Sat}_{\phi}(\Delta', l)$ and $\mathsf{Sat}_{\phi}(|\Delta|, l^{\perp}) = \mathsf{Sat}_{\phi}(\Delta', l^{\perp})$. The two new leaves are tagged as open.

- Backtrack: $\Delta_1, l^d, \Delta_2 \| \phi, C \Rightarrow \Delta_1, l^{\perp} \| \phi, C$ if $\Delta_1, l, \Delta_2 \models \neg C$ and no decision literal is in Δ_2.

  Let π_1 be a partial proof-tree corresponding to $\Delta_1, l^d, \Delta_2 \| \phi, C$. The open leaf corresponding to $|\Delta_1, l^d, \Delta_2|$ is of the form $\Delta', \phi', C' \vdash^{|\Delta_1|, l, |\Delta_2|}$ where ϕ' represents ϕ, C' represents C and $\mathsf{Sat}_{\phi}(\Delta_1, l, \Delta_2) = \mathsf{Sat}_{\phi}(\Delta')$. Let $C = l_1 \vee \ldots \vee l_n$.

  From $|\Delta_1|, l, |\Delta_2| \models \neg C$ we get $\forall i, l_i^{\perp} \in |\Delta_1|, l, |\Delta_2| \subseteq \mathsf{Sat}_{\phi, C}(|\Delta_1|, l, |\Delta_2|) = \mathsf{Sat}_{\phi, C}(\Delta')$.

  We extend π_1 into π_2 by replacing this leaf by the following (complete) proof-tree:

  $$\frac{\left(\begin{array}{c}\overline{\Delta', \phi', C' \vdash^{|\Delta_1|, l, |\Delta_2|} [l_i^{\perp}]}\end{array}\right)_{l_i \in \mathsf{lit}(C)} \quad \vdots \wedge^{+}. \quad \Delta', \phi', C' \vdash^{|\Delta_1|, l, |\Delta_2|} [C'^{\perp}]}{\Delta', \phi', C' \vdash^{|\Delta_1|, l, |\Delta_2|}}$$

The top rules can be applied since $l_i^{\perp} \in |\Delta_1|, l, |\Delta_2|$ and $\mathsf{lit\_ctxt}(\Delta', \phi', C'), l_i \models_{\mathcal{T}}$.

Noticing that $[\![\Delta_1, l^d, \Delta_2]\!] = [\![\Delta_1]\!] \cup \{(|\Delta_1|, l^{\perp})\}$, the open leaves of π_2 are in 1-to-1 correspondence with $[\![\Delta_1]\!] \cup \{(|\Delta_1|, l^{\perp})\}$, so π_2 is a $\sharp(C')+1$-extension of π_1 that corresponds to $\Delta_1, l^{\perp} \| \phi, C$ (and note that $\sharp(C') \leq 2\sharp(\phi)$).

- Unit propagation : $\Delta \| \phi, C \vee l \Rightarrow \Delta, l \| \phi, C \vee l$ where $\Delta \models \neg C$, $l \notin \Delta$, $l^{\perp} \notin \Delta$.

  Let π_1 be a partial proof-tree corresponding to $\Delta \| \phi, C \vee l$. The open leaf corresponding to $|\Delta|$ is of the form $\Delta', \phi', C' \vdash^{|\Delta|}$ where ϕ' represents ϕ, C' represents $C \vee l$ and $\mathsf{Sat}_{\phi, C \vee l}(|\Delta|) = \mathsf{Sat}_{\phi, C \vee l}(\Delta')$. Let $C = l_1 \vee \ldots \vee l_n$. From $\Delta \models \neg C$ we get $\forall i$, $l_i^{\perp} \in |\Delta| \subseteq \mathsf{Sat}_{\phi, C \vee l}(|\Delta|) = \mathsf{Sat}_{\phi, C \vee l}(\Delta')$.

  We extend π_1 into π_2 by replacing this leaf by the following (partial) proof-tree:

$$\dfrac{\dfrac{\dfrac{\Delta', l, \phi', C' \vdash^{|\Delta|, l}}{\Delta', \phi', C' \vdash^{|\Delta|, l} l^{\perp}}}{\Delta', \phi', C' \vdash^{|\Delta|, l} [l^{\perp}]} \quad \left(\dfrac{}{\Delta', \phi', C' \vdash^{|\Delta|, l} [l_i^{\perp}]} \right)_{l_i \in \mathsf{lit}(C)} \\ \vdots \wedge^{+}. \\ \dfrac{\dfrac{\Delta', \phi', C' \vdash^{|\Delta|, l} [C'^{\perp}]}{\Delta', \phi', C' \vdash^{|\Delta|, l}}}{\Delta', \phi', C' \vdash^{|\Delta|}}$$

  The top-right rules can be applied since $l_i^{\perp} \in |\Delta|$ and $\mathsf{lit\_ctxt}(\Delta', \phi', C'), l_i \models_{\mathcal{T}}$ and the new leaves are closed. The top-left leaf is tagged as open.

  Noticing that $[\![\Delta, l]\!] \cup \{|\Delta, l|\} = [\![\Delta]\!] \cup \{(|\Delta|, l)\}$, the open leaves of π_2 are in 1-to-1 correspondence with $[\![\Delta, l]\!] \cup \{|\Delta, l|\}$, so π_2 is a $\sharp(C')+3$-extension of π_1 that corresponds to $\Delta, l \| \phi, C \vee l$ (and note that $\sharp(C') \leq 2\sharp(\phi)$).

- Theory Propagate: $\Delta \| \phi \Rightarrow \Delta, l \| \phi$ where $l \in \mathsf{Sat}_{\phi}(\Delta)$ and $l \notin \Delta, l^{\perp} \notin \Delta$.

  Let π_1 be a partial proof-tree corresponding to $\Delta \| \phi$. The open leaf corresponding to $|\Delta|$ is of the form $\Delta', \phi' \vdash^{|\Delta|}$ where ϕ' represents ϕ and $\mathsf{Sat}_{\phi}(\Delta) = \mathsf{Sat}_{\phi}(\Delta')$. We extend π_1 into π_2 by replacing this leaf by the following (partial) proof-tree:

$$\dfrac{\Delta', \phi' \vdash^{|\Delta|, l}}{\Delta', \phi' \vdash^{|\Delta|}}$$

  Noticing that $\mathsf{Sat}_{\phi}(\Delta) = \mathsf{Sat}_{\phi}(\Delta, l)$, π_2 is a 1-extension of π_1 that corresponds to $\Delta, l \| \phi$.

Now, we have said that the aim of $\mathsf{DPLL}(\mathcal{T})$ was to prove the inconsistency of a set of clauses ϕ with respect to a theory. Strictly speaking, we have not defined what that meant yet, because the notions of literal and inconsistency modulo theory have been taken as abstract notions (and inconsistency is a notion pertaining to sets of literals, not clauses).

In order to say that $\mathsf{DPLL}(\mathcal{T})$, and then by simulation $\mathsf{LK}^p(\mathcal{T})$, fulfil their aim, we need to refer to an actual instance of those abstract notions:

Definition 4.5 (Concrete literals and concrete inconsistency modulo theory) *We fix a signature (a collection of function symbols and predicate symbols together with their arities) and we consider a theory $\mathcal{T}$ on this signature, given as a set of closed first formulas on this signature.*

Literals *are closed atoms or negations of closed atoms on this signature, with the obvious notion of involutive negation.*

The property that a set $\mathcal{P}$ of atoms is inconsistent modulo theory ($\mathcal{P} \models_{\mathcal{T}}$) is a particular case of the property that a set ϕ of clauses is inconsistent modulo theory*, denoted $\phi \models_{\mathcal{T}}$: there is no (first-order) model of $\mathcal{T}$ that is a model of every clause in ϕ (seen as a disjunction).*

This itself is a particular case of the property that ϕ semantically entails *a clause C (which can be taken to be the empty clause $\bot$), denoted $\phi \models_{\mathcal{T}} C$: every (first-order) model of $\mathcal{T}$ that is a model of every clause in ϕ, is also a model of C. Similarly when A is a formula of polarised classical logic, we write $\phi \models_{\mathcal{T}} A$ when every model of $\mathcal{T}$ that is a model of every clause in ϕ, is also a model of A (positive and negative conjunctions are identified, positive and negative disjunctions are identified).*

Remark 4.5 *Notice that, for such a concrete inconsistency predicate, if $\Delta, l \models_{\mathcal{T}}$ and $\Delta, l^{\perp} \models_{\mathcal{T}}$ then $\Delta \models_{\mathcal{T}}$ (we will use this later).*

Corollary 4.2 (Completeness) *If $\phi \models_{\mathcal{T}}$ and ϕ' represents ϕ then there is a complete derivation in $\mathsf{LK}^p(\mathcal{T})$ of $\phi' \vdash$.*

Proof. By completeness of basic $\mathsf{DPLL}(\mathcal{T})$ [NOT05, NOT06] and Theorem 4.1.

Now the point of having mentioned quantitative information in Theorem 4.1, via the notion of n-extension, is to motivate the idea that performing proof-search directly in $\mathsf{LK}^p(\mathcal{T})$ is not less efficient than running $\mathsf{DPLL}(\mathcal{T})$: in the above Corollary, the final size of the proof-tree is bounded by a linear function of the length of the $\mathsf{DPLL}(\mathcal{T})$ run (and the proportionality ratio is itself an affine function of the size of the original problem).

Now we also need to make sure that this final proof-tree is indeed found as efficiently as running $\mathsf{DPLL}(\mathcal{T})$, which would not be the case if the proof-search mechanisms of sequent calculus offered a search-space that is much wider than in $\mathsf{DPLL}(\mathcal{T})$. For this we identify a proof-search strategy, by looking at the proof of Theorem 4.1. The strategy imposes that proof-search follow those rules:

- In order to place a formula in focus, all of its literals must be polarised and at most one of them has a negative polarity; if such a literal exists it must have been polarised in the rule just under the focusing rule.
- Only polarise literals positively when they are semantically entailed by the left-hand side of the sequent.
- Always follow an analytic cut by polarising the cut-literal above each premiss.

Those conditions identify the complete proof-trees that are the images of $\mathsf{DPLL}(\mathcal{T})$ runs via the simulation.

5 Extending $\mathsf{DPLL}(\mathcal{T})$ with Backjump and Lemma learning

5.1 Presentation of Backjump and Lemma learning in $\mathsf{DPLL}(\mathcal{T})$

We now consider $\mathsf{DPLL}_{bj}(\mathcal{T})$, a more advanced version of $\mathsf{DPLL}(\mathcal{T})$, which involves backjumping and lemma learning features. The $\mathsf{DPLL}_{bj}(\mathcal{T})$ transition system extends basic $\mathsf{DPLL}(\mathcal{T})$ with the rules known as $\mathcal{T}$-Backjump, $\mathcal{T}$-Learn, $\mathcal{T}$-Forget, and Restart. The implementation of these rules drastically increase the efficiency of SMT-solvers. Here again we closely follow the presentation of [NOT06].

Definition 5.1 ($\mathsf{DPLL}_{bj}(\mathcal{T})$) *The states of the $\mathsf{DPLL}_{bj}(\mathcal{T})$ transition system are the same as the ones of $\mathsf{DPLL}_{bj}(\mathcal{T})$ and its rules extend the ones of $\mathsf{DPLL}(\mathcal{T})$ (see Definition 3.5) with the following ones:*

- *$\mathcal{T}$-Backjump:* $\Delta_1, l^d, \Delta_2 \| \phi, C \Rightarrow \Delta_1, l_{bj} \| \phi, C$ *when*
  1. $\Delta_1, l^d, \Delta_2 \models \neg C$.
  2. $\Delta_1 \models \neg C_0$
  3. $\phi, C \models_{\mathcal{T}} C_0 \vee l_{bj}$
  4. $l_{bj} \notin \Delta_1$, $l_{bj}^{\perp} \notin \Delta_1$ *and* $l_{bj} \in \mathit{lit}(\phi, \Delta_1, l^d, \Delta_2)$.

  for some clause C_0 such that $\mathit{lit}(C_0) \subseteq \mathit{lit}(\phi, C)$.
- *$\mathcal{T}$-Learn:* $\Delta \| \phi \Rightarrow \Delta \| \phi, C$ *if* $\mathit{lit}(C) \subseteq \mathit{lit}(\phi)$ *and* $\phi \models_{\mathcal{T}} C$.
- *$\mathcal{T}$-Forget:* $\Delta \| \phi, C \Rightarrow \Delta \| \phi$ *if* $\phi \models_{\mathcal{T}} C$.
- *Restart:* $\Delta \| \phi \Rightarrow \emptyset \| \phi$.

In rule $\mathcal{T}$-Backjump, $C_0 \vee l_{bj}$ is called the *backjump clause*.

5.2 Extension of $\mathsf{LK}^p(\mathcal{T})$ with cuts

In the rules $\mathcal{T}$-Backjump and $\mathcal{T}$-Learn, we see that a new clause is used (e.g. in the side-conditions) that we had not seen before (respectively: $C_0 \vee l_{bj}$ and C). In order to simulate those extra rules in $\mathsf{LK}^p(\mathcal{T})$, we need to extend the calculus with a general cut rule, so that the production of the new clause corresponds to the choice of the cut-formula.

Definition 5.2 ($\mathsf{LK}^p(\mathcal{T})$ with cut) *System $\mathsf{LK}^p_c(\mathcal{T})$ is obtained by extending system $\mathsf{LK}^p(\mathcal{T})$ with the following cut-rule:*

$$\frac{\Gamma, l_1, \ldots, l_n \vdash^{\mathcal{P}} \qquad \Gamma, (l_1^{\perp} \vee^{-} \ldots \vee^{-} l_n^{\perp}) \vdash^{\mathcal{P}}}{\Gamma \vdash^{\mathcal{P}}} \; \mathit{cut}$$

We define the size of proof-trees in $\mathsf{LK}^p_c(\mathcal{T})$ as we did for $\mathsf{LK}^p(\mathcal{T})$ but ignoring the left-branch of the cut-rules. As we shall see in the simulation theorem, this definition mimicks the fact that the length of $\mathsf{DPLL}(\mathcal{T})$ sequences is a complexity measure that ignores the cost of checking the side-conditions.

5.3 Simulation

As opposed to what happens in a basic $\mathsf{DPLL}(\mathcal{T})$ run, the extra rules of $\mathsf{DPLL}_{bj}(\mathcal{T})$ can add or remove objects from a state (clauses to falsify, literals). On the contrary, once such an object is introduced in a $\mathsf{LK}^p(\mathcal{T})$ sequent by the proof-search process, this data persists in the entire subtree proving the sequent. This phenomenon is described in [NOT05] who conclude that an abstract presentation of $\mathsf{DPLL}(\mathcal{T})$ based on sequent calculus is necessarily too rigid to model the rules that practical implementations of $\mathsf{DPLL}(\mathcal{T})$ rely on. The simulation theorem we propose in this section shows that a combination of tags and polarisation can actually overcome this discrepancy. Such a simulation theorem for $\mathsf{DPLL}_{bj}(\mathcal{T})$ however requires to slightly relax the notion of correspondance between states and partial proof-trees in $\mathsf{LK}^p(\mathcal{T})$: we should allow the sequent label of an open leaf to contain some objects that have disappeared from the corresponding state.

Definition 5.3 (Correspondence between states and $\mathsf{LK}^p_c(\mathcal{T})$ partial proof-trees)

A partial proof-tree π corresponds to a state $\Delta \| \phi$ if

- *there is a mapping from the open leaves of π to the elements of $[\![\Delta]\!] \cup \{|\Delta|\}$;*
- *the sequent label of an open leaf mapped to a set $\Delta_0 \in [\![\Delta]\!] \cup \{|\Delta|\}$ is of the form $\Delta', \phi', \Gamma \vdash^{\mathcal{P}}$, where:*
  - $\Delta_0 \subseteq \mathcal{P} \subseteq \mathit{Sat}(\Delta')$.
  - *ϕ' represents ϕ,*
  - $\forall A \in \Gamma$, $\phi \models A$

A partial proof-tree π corresponds to *the state UNSAT if it has no open leaf.*

The difference with the previous notion of correspondence is that the open leaves are no longer in 1-to-1 correspondence with the elements of $[\![\Delta]\!] \cup \{|\Delta|\}$: for any $\Delta_0 \in [\![\Delta]\!] \cup \{|\Delta|\}$, zero, one, or several open leaves may correspond to it.

Furthermore, the sequent $\Delta', \phi', \Gamma \vdash^{\mathcal{P}}$ labelling such a leaf correspnding to Δ_0 may

- declare more positive literals than Δ_0
- have Δ' entail more literals than Δ_0
- contain extra formulae Γ in its antecedent, that are not representing clauses in the $\mathsf{DPLL}(\mathcal{T})$ state $\Delta \| \phi$.

As sometimes several leaves correspond to the same $\Delta_0 \in [\![\Delta]\!] \cup \{|\Delta|\}$, we need to simultaneously replace those leaves in our process of extending partial proof-trees. This notion is formalised as follows:

Definition 5.4 (Parallel n-extension of partial proof-trees)

- *Let $\mathcal{F} = (s_a)_{a \in A}$ be a family of sequents. An n-action over $\mathcal{F}$ is a family $(\pi_a)_{a \in A}$ of partial proof-trees in $\mathsf{LK}^p_c(\mathcal{T})$ such that*
  - *the conclusion (root) of π_a is s_a for all $a \in A$;*

– *the family* $(\pi_a)_{a\in A}$ *can be generated by some partial proof-trees whose total size is less than* n.

- π_2 *is a* parallel n-extension *of* π_1 *according to* $(\pi_a)_{a\in A}$ *if*
  – $(\pi_a)_{a\in A}$ *is an* n*-action over the family of sequents labelling the open leaves of* π_1
  – π_2 *is obtained from* π_1 *by replacing every open leaf of* π_1 *whose label is* s_a*, by* π_a.

We can now formulate and prove the equivalent of Theorem 4.1 for $\mathsf{DPLL}_{bj}(\mathcal{T})$. Since $\mathsf{DPLL}_{bj}(\mathcal{T})$ extends $\mathsf{DPLL}(\mathcal{T})$, we still have to simulate Fail, Decide, Unit Propagation, Theory Propagation and we could argue that we have already dealt with those transitions in the proof of Theorem 4.1. However that theorem involves a different notion of correspondence between states and partial proof-trees and, while this notion is stronger, Theorem 4.1 does not strictly speaking entail what we need here, i.e. :

Theorem 5.1 (Simulation of $\mathsf{DPLL}(\mathcal{T})$ in $\mathsf{LK}^p_{\mathsf{c}}(\mathcal{T})$) *If* $\Delta\|\phi \Rightarrow_{DPLL(\mathcal{T})} \mathcal{S}_2$ *and* π_1 *corresponds to* $\Delta\|\phi$*, there is a parallel* $(4^\sharp_\sharp(\phi)+6)$*-extension* π_2 *of* π_1 *such that* π_2 *corresponds to the state* $\mathcal{S}_2$.

Proof. The proof can easily be adapted from that of Theorem 4.1; see Appendix A.

Corollary 5.2 *If* $\phi \models_{\mathcal{T}}$*, then there is a complete derivation in* $LK^p_{\mathsf{c}}(\mathcal{T})$ *of* $\Delta', \phi', \Gamma \vdash^{\mathcal{P}}$ *, provided* ϕ' *corresponds to* ϕ*,* $\mathcal{P} \subseteq \mathsf{Sat}(\Delta')$*, and* $\phi \models_{\mathcal{T}} \Gamma$.

Proof. By completeness of basic $\mathsf{DPLL}(\mathcal{T})$ and Theorem 5.1.

Theorem 5.3 (Simulation of $\mathsf{DPLL}_{bj}(\mathcal{T})$ in $\mathsf{LK}^p_{\mathsf{c}}(\mathcal{T})$) *If* $\Delta\|\phi \Rightarrow_{DPLL_{bj}(\mathcal{T})} \mathcal{S}_2$ *and* π_1 *corresponds to* $\Delta\|\phi$*, there is a parallel* $(4^\sharp_\sharp(\phi)+6)$*-extension* π_2 *of* π_1 *such that* π_2 *corresponds to the state* $\mathcal{S}_2$.

Proof. Now we can concentrate on simulating (in $\mathsf{LK}^p_{\mathsf{c}}(\mathcal{T})$) the new rules:

$\mathcal{T}$-Backjump $\Delta_1, l^d, \Delta_2\|\phi, C \Rightarrow \Delta_1, l_{bj}\|\phi, C$ with

1. $\Delta_1, l^d, \Delta_2 \models \neg C$.
2. $\Delta_1 \models \neg C_0$
3. $\phi, C \models_{\mathcal{T}} C_0 \vee l_{bj}$
4. $l_{bj} \notin \Delta_1$, $l^\perp_{bj} \notin \Delta_1$ and $l_{bj} \in \mathsf{lit}(\phi, \Delta_1, l^d, \Delta_2)$.

for some clause $C_0 = l_1 \vee \ldots \vee l_n$ such that $\mathsf{lit}(C_0) \subseteq \mathsf{lit}(\phi, C)$.

Let π_1 be a partial proof-tree corresponding to $\Delta_1, l^d, \Delta_2\|\phi, C$. We have to build a π_2 that corresponds to $\Delta_1, l_{bj}\|\phi, C$.

Notice that if $\Delta_0 \in [\![\Delta_1, l^d, \Delta_2]\!] \cup \{|\Delta_1, l^d, \Delta_2|\}$, then either $\Delta_0 \in [\![\Delta_1]\!]$ or $|\Delta_1| \subseteq \Delta_0$.

We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a\in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

– for a sequent s_a labelling an open leaf mapped to $\Delta_0 \in [\![\Delta_1]\!]$, we define π_a as a 1-node tree labelled with s_a

– for a sequent $s_a = \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf mapped to Δ_0 containing $|\Delta_1|$ as a subset, then

∗ If $l_{bj} \notin \mathcal{P}$ and $l_{bj}^{\perp} \notin \mathcal{P}$, we define π_a as

$$\dfrac{\dfrac{\dfrac{\Delta', l_{bj}, \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}}}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}} l_{bj}^{\perp}}}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}} [l_{bj}^{\perp}]} \qquad \left(\dfrac{}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}} [l_i^{\perp}]} \right)_{l_i \in \mathsf{lit}(C_0)} \vdots \wedge^{+}. \qquad \dfrac{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}} [{C_0'}^{\perp}]}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}}}}{\dfrac{\Delta', \phi', {C_0'}^{\perp}, C', \Gamma \vdash^{\mathcal{P}, l_{bj}} \qquad \Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}, l_{bj}}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l_{bj}}}\ \mathsf{cut} \text{ on } C_0'}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}$$

where the cut-formula C_0' represents $C_0 \vee l_{bj}$.
The left premiss is closed by applying Corollary 5.2 on the hypothesis $\phi, C \models_{\mathcal{T}} C_0 \vee l_{bj}$, i.e. $\phi, C, C_0^{\perp}, l_{bj}^{\perp} \models_{\mathcal{T}}$: we get a complete proof-tree of $\Delta', \phi', {C_0'}^{\perp}, C', \Gamma \vdash^{\mathcal{P}, l_{bj}}$. For the top-right branches, hypothesis $|\Delta_1| \models \neg C_0$ entails that for all $l_i \in \mathsf{lit}(C_0)$, $l_i^{\perp} \in |\Delta_1| \subseteq \Delta_0 \subseteq \mathcal{P}$, and then the leaves are closed as $\mathsf{lit\_ctxt}(\Delta', \phi', C_0', C', \Gamma) \models_{\mathcal{T}} l_i^{\perp}$.

∗ If $l_{bj} \in \mathcal{P}$, we define π_a as a 1-node tree labelled with s_a, but the leaf is now mapped to $|\Delta_1, l_{bj}|$.

∗ If $l_{bj}^{\perp} \in \mathcal{P}$, we close the whole branch altogether by defining π_a as the following complete proof-tree:

$$\dfrac{\dfrac{}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}} [l_{bj}^{\perp}]} \qquad \left(\dfrac{}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}} [l_i^{\perp}]} \right)_{l_i \in \mathsf{lit}(C_0)} \vdots \wedge^{+}. \qquad \dfrac{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}} [{C_0'}^{\perp}]}{\Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}}}}{\dfrac{\Delta', \phi', {C_0'}^{\perp}, C', \Gamma \vdash^{\mathcal{P}} \qquad \Delta', \phi', C_0', C', \Gamma \vdash^{\mathcal{P}}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}\ \mathsf{cut} \text{ on } C_0'}$$

We argue that $(\pi_a)_{a \in A}$ is a $(4\sharp(\phi) + 6)$-action: the family can be described by the two trees above; now remember that $\sharp(C_o')$ is less than $2\sharp(\phi)$, and therefore the two decomposition phases above have at most $2\sharp(\phi)$ steps each; finally there are 6 other steps (remember we are not counting the left-premisses of cuts).[1]

$\mathcal{T}$-Learn $\Delta \| \phi \Rightarrow \Delta \| \phi, C$ if $\mathsf{lit}(C) \subseteq \mathsf{lit}(\phi, \Delta)$ and $\phi \models_{\mathcal{T}} C$.

[1] We cannot anticipate the size of the proof-tree closing the left premiss, and we therefore ignore left premisses of cuts to compute the size of trees. Notice that, similarly, the length of the $\mathsf{DPLL}_{bj}(\mathcal{T})$ run ignores the cost of checking the side-condition as well.

Let π_1 be a partial proof-tree corresponding to $\Delta\|\phi$. We have to build π_2 that corresponds to $\Delta\|\phi, C$.

We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a\in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

Let s_a be a sequent labelling an open leaf; it is of the form $\Delta', \phi', \Gamma \vdash^{\mathcal{P}}$; let C' represent C and let π_a be

$$\frac{\Delta', \phi', C'^{\perp}, \Gamma \vdash^{\mathcal{P}} \qquad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}{\Delta', \phi', \Gamma \vdash^{\mathcal{P}}}\ \mathsf{cut} \text{ on } C'$$

Again, the left-premiss is closed by applying Corollary 5.2 on the hypothesis $\phi \models_{\mathcal{T}} C$, i.e. $\phi, C^{\perp} \models_{\mathcal{T}}$: we get a complete proof-tree of $\Delta', \phi', C'^{\perp}, \Gamma \vdash^{\mathcal{P}}$.

We argue that $(\pi_a)_{a\in A}$ is a 1-action.

$\mathcal{T}$-Forget $\Delta\|\phi, C \Rightarrow \Delta\|\phi$ if $\phi \models_{\mathcal{T}} C$.

Let π_1 be a partial proof-tree corresponding to $\Delta\|\phi$. We take $\pi_2 := \pi_1$ (a parallel 0-extension). It corresponds to $\Delta\|\phi$ since, in a sequent $\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf, we can consider C', Γ as the new Γ (knowing that $\phi \models_{\mathcal{T}} C$).

Restart: $\Delta\|\phi \Rightarrow \emptyset\|\phi$

Similarly, take $\pi_2 := \pi_1$.

Again, we have mentioned quantitative information in the simulation just to emphasize that the size of the proof-trees produced are proportional to the length of the advanced $\mathsf{DPLL}(\mathcal{T})$ runs, and the proportionality ratio is the size needed to describe actions. We argued that this ratio is therefore an affine function of the size of the original problem, at least if actions are described with some sharing (unnecessary for basic $\mathsf{DPLL}(\mathcal{T})$) since one action may transform many leaves in parallel (but in very similar ways).

At this point, we should also look into defining a proof-search strategy identifying those complete proof-trees that are the images of advanced $\mathsf{DPLL}(\mathcal{T})$ runs, just as we did with basic $\mathsf{DPLL}(\mathcal{T})$. This is left for a future paper.

6 Conclusion and Further work

In this paper we have shown how to express the $\mathsf{DPLL}(\mathcal{T})$ procedure as the incremental construction of proof-trees in sequent calculus. For this we used the focused sequent calculus $\mathsf{LK}^p(\mathcal{T})$ for polarised classical logic, which is able to restrict the search space compared to Gentzen's sequent calculus.

We simulated both the basic $\mathsf{DPLL}(\mathcal{T})$ procedure and the advanced $\mathsf{DPLL}(\mathcal{T})$ procedure with backjump, lemma learning, etc, Our simulations revealed that the former needs analytic cuts while the latter needs general cuts. Moreover in backjump, the backjump clause corresponds to the cut-formula in sequent calculus, so the cleverness that goes into finding a backjump clause translates as the cleverness that goes into picking a good cut-formula: neither task is described by the procedures.

Our work relates to the inference systems described in e.g. [Tin02]. Our point here is to avoid designing an inference system taylored to the simulation of $\mathsf{DPLL}(\mathcal{T})$, but rather to stick to traditional presentations of sequent calculus (notwithstanding the use of polarisation, focusing, and calls to a decision procedure specific to the theory): for instance, we still take formulae to be trees and inference rules to organise the root-first decomposition of their connectives, rather than using $\mathsf{DPLL}(\mathcal{T})$'s more flexible structures. Our hope by sticking to traditional presentations of sequent calculus is that the same framework can be reused to import other techniques from automated reasoning, which is our next programme of research.

In fact, we also used polarities and focusing to simulate the system of [Tin02], and we plan to investigate this line of research further hoping to incorporate the recent work extending $\mathsf{DPLL}(\mathcal{T})$ with e.g. full first-order logic and/or equality [Bau00, BT08, BT11]. The full version of $\mathsf{LK}^p(\mathcal{T})$ is indeed designed for handling quantifiers and equalities, so we hope to relate it to other techniques such as unification, paramodulation and superposition, etc.

References

[AFG+11] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to Coq through proof witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Proc. of the 1st Int. Conf. on Certified Programs and Proofs (CPP'11)*, volume 7086 of *LNCS*, pages 135–150. Springer, December 2011.

[And92] J. M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic Comput.*, 2(3):297–347, 1992.

[Bau00] Peter Baumgartner. FDPLL - a first order Davis-Putnam-Longeman-Loveland procedure. In David A. McAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE'00)*, volume 1831 of *LNCS*, pages 200–219. Springer-Verlag, June 2000.

[BNOT06] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT Modulo Theories. In Miki Hermann and Andrei Voronkov, editors, *Proc. of the the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 512–526. Springer-Verlag, November 2006.

[BT08] Peter Baumgartner and Cesare Tinelli. The model evolution calculus as a first-order DPLL method. *Artificial Intelligence*, 172(4-5):591–632, 2008.

[BT11] Peter Baumgartner and Cesare Tinelli. Model evolution with equality modulo built-in theories. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Proc. of the 23rd Int. Conf. on Automated Deduction (CADE'11)*, volume 6803 of *LNCS*, pages 85–100. Springer-Verlag, July 2011.

[Coq] The Coq Proof Assistant.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. of the ACM Press*, 7(3):201–215, 1960.

[Gir87] Jean-Yves Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: Classical logic. *Math. Structures in Comput. Sci.*, 1(3):255–296, 1991.

[Lau03] Olivier Laurent. Polarized proof-nets and lambda-mu calculus. *Theoret. Comput. Sci.*, 1(290):161–188, 2003.

[LC09] Stéphane Lescuyer and Sylvain Conchon. Improving Coq propositional reasoning using a lazy CNF conversion scheme. In *Proc. of the 7th Int. Conf. on Frontiers of combining systems (FroCoS'09)*, pages 287–303, Berlin, Heidelberg, 2009. Springer-Verlag.

[LM09] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoret. Comput. Sci.*, 410(46):4747–4768, 2009.

[MN07] Dale Miller and Vivek Nigam. Incorporating tables into proofs. In Jacques Duparc and Thomas A. Henzinger, editors, *Proc. of the 16th Annual Conf. of the European Association for Computer Science Logic (CSL'07)*, volume 4646 of *LNCS*, pages 466–480. Springer-Verlag, September 2007.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51:125–157, 1991.

[NOT05] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and abstract DPLL Modulo Theories. In Franz Baader and Andrei Voronkov, editors, *Proc. of the the 11th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)*, volume 3452 of *LNCS*, pages 36–50. Springer-Verlag, March 2005.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. of the ACM Press*, 53(6):937–977, 2006.

[Tin02] Cesare Tinelli. A DPLL-based calculus for ground satisfiability modulo theories. In Giovambattista Ianni and Sergio Flesca, editors, *Proc. of the 8th European Conf. on Logics in Artificial Intelligence*, volume 2424 of *LNAI*, pages 308–319. Springer-Verlag, 2002.

A Full proofs

Theorem 5.1 (Simulation of $\mathsf{DPLL}(\mathcal{T})$ in $\mathsf{LK}^p_{\mathsf{c}}(\mathcal{T})$) *If $\Delta\|\phi \Rightarrow_{DPLL(\mathcal{T})} \mathcal{S}_2$ and π_1 corresponds to $\Delta\|\phi$, there is a parallel $(4\sharp_\sharp(\phi)+6)$-extension π_2 of π_1 such that π_2 corresponds to the state $\mathcal{S}_2$.*

Proof. By case analysis on the nature of the transition:

- Fail: $\Delta\|\phi, C \Rightarrow \mathsf{UNSAT}$ with $\Delta \models \neg C$ and there is no decision literal in Δ.
- Backtrack: $\Delta_1, l^d, \Delta_2\|\phi, C \Rightarrow \Delta_1, l^\perp\|\phi, C$
  with $\Delta_1, l, \Delta_2 \models \neg C$ and there is no decision literal in Δ_2.

  We treat Fail and Backtrack at the same time, taking $\Delta := \Delta_1, l, \Delta_2$ in the case of Backtrack.

  Let π_1 be a partial proof-tree corresponding to $\Delta\|\phi, C$.

  We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a\in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

  - for a sequent s_a labelling an open leaf mapped to $\Delta_0 \in [\![\Delta]\!]$, we define π_a as a 1-node tree labelled with s_a
  - for a sequent $s_a = \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf mapped to $|\Delta|$, we define π_a as

$$\dfrac{\left(\dfrac{}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}} [l_i^\perp]}\right)_{l_i \in \mathsf{lit}(C)} \quad \vdots \wedge^+. \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}} [C'^\perp]}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}$$

  For the top rules, hypothesis $\Delta \models \neg C$ entails that for all $l_i \in \mathsf{lit}(C)$, $l_i^\perp \in |\Delta| \subseteq \mathcal{P} \subseteq \mathsf{Sat}_{\phi,C}(\Delta')$, so $l_i^\perp$ is $\mathcal{P}$-positive and $\mathsf{lit\_ctxt}(\Delta', \phi', C', \Gamma, l_i) \models_{\mathcal{T}}$.

  π_2 is a $(\sharp(C')+1)$-extension of π_1 (and note that $\sharp(C') \leq 2\sharp(\phi)$).

  Moreover in the case of Fail, since there are no decision literals in Δ ($[\![\Delta]\!] = \emptyset$), the open leaves of π_1 are all mapped to $|\Delta|$ and therefore π_2 is complete and corresponds to the UNSAT state of the $\mathsf{DPLL}(\mathcal{T})$ run. In the case of Backtrack, the open leaves remaining in π_2 are those open leaves of π_1 mapped to $[\![\Delta_1, l^d, \Delta_2]\!] = [\![\Delta_1]\!] \cup \{|\Delta_1, l^\perp|\}$, and therefore π_2 corresponds to the state $\Delta_1, l^\perp\|\phi, C$.

- Decide: $\Delta\|\phi \Rightarrow \Delta, l^d\|\phi$ where $l \notin \Delta$, $l^\perp \notin \Delta$, $l \in \mathsf{lit}(\phi)$.

  Let π_1 be a partial proof-tree corresponding to $\Delta\|\phi$.

  We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a\in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

– for a sequent s_a labelling an open leaf mapped to $\Delta_0 \in [\![\Delta]\!]$, we define π_a as a 1-node tree labelled with s_a

– for a sequent $s_a = \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf mapped to $|\Delta|$, then:

∗ If neither $l \in \mathcal{P}$ nor $l^\perp \in \mathcal{P}$, we define π_a as

$$\dfrac{\dfrac{\Delta', l^\perp, \phi', \Gamma \vdash^{\mathcal{P}, l^\perp}}{\Delta', l^\perp, \phi', \Gamma \vdash^{\mathcal{P}}} \qquad \dfrac{\Delta', l^\perp, \phi', \Gamma \vdash^{\mathcal{P}, l}}{\Delta', l, \phi', \Gamma \vdash^{\mathcal{P}}}}{\Delta', \phi', \Gamma \vdash^{\mathcal{P}}}$$

Note that we use here the analytic cut rule of $\mathsf{LK}^p(\mathcal{T})$.

∗ If $l \in \mathcal{P}$ (resp. $l^\perp \in \mathcal{P}$) then we define π_a as a 1-node tree labelled with s_a, and the open leaf is now mapped to $|\Delta, l| \in [\![\Delta, l^d]\!] \cup \{|\Delta, l^d|\}$ (resp. $(|\Delta, l^\perp| \in [\![\Delta, l^d]\!] \cup \{|\Delta, l^d|\})$.

π_2 is a 3-extension of π_1 that corresponds to $\Delta, l^d \| \phi$.

- Unit propagation : $\Delta \| \phi, C \vee l \Rightarrow \Delta, l \| \phi, C \vee l$ with $\Delta \models \neg C$, $l \notin \Delta$, $l^\perp \notin \Delta$.

Let π_1 be a partial proof-tree corresponding to $\Delta \| \phi, C \vee l$.

We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a \in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

– for a sequent s_a labelling an open leaf mapped to $\Delta_0 \in [\![\Delta]\!]$, we define π_a as a 1-node tree labelled with s_a

– for a sequent $s_a = \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf mapped to $|\Delta|$ (C' now represents $C \vee l$), then:

∗ If neither $l \in \mathcal{P}$ nor $l^\perp \in \mathcal{P}$, we define π_a as

$$\dfrac{\dfrac{\dfrac{\Delta', l, \phi', C', \Gamma \vdash^{\mathcal{P}, l}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l}\ l^\perp}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l}\ [l^\perp]} \qquad \left(\dfrac{}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l}\ [l_i^\perp]} \right)_{l_i \in \mathsf{lit}(C)}}{\vdots \wedge^+.}}{\dfrac{\dfrac{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l}\ [C'^\perp]}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}, l}}}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}}$$

For the top-right rules, hypothesis $\Delta \models \neg C$ entails that for all $l_i \in \mathsf{lit}(C)$, $l_i^\perp \in |\Delta| \subseteq \mathcal{P} \subseteq \mathsf{Sat}_{\phi,C}(\Delta')$, so $l_i^\perp$ is $\mathcal{P}$-positive and $\mathsf{lit\_ctxt}(\Delta', \phi', C', \Gamma, l_i) \models_{\mathcal{T}}$. The top-left leaf is tagged as open.

∗ If $l \in \mathcal{P}$ then we define π_a as a 1-node tree labelled with s_a, and the open leaf is now mapped to $|\Delta, l|$.

∗ If $l^\perp \in \mathcal{P}$ then we close the branch altogether by defining π_a as:

$$\frac{\overline{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}} [l^{\perp}]} \qquad \left(\overline{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}} [l_i^{\perp}]} \right)_{l_i \in \mathsf{lit}(C)} \quad \vdots \wedge^{+}. \quad \Delta', \phi', C', \Gamma \vdash^{\mathcal{P}} [C'^{\perp}]}{\Delta', \phi', C', \Gamma \vdash^{\mathcal{P}}}$$

We argue that π_2 is a $(2\sharp(C') + 4)$-extension of π_1 that corresponds to $\Delta, l \| \phi, C \vee l$ (and note that $\sharp(C') \leq 2\sharp(\phi)$).

- Theory Propagate: $\Delta \| \phi \Rightarrow \Delta, l \| \phi$ where $l \in \mathsf{Sat}_{\phi}(\Delta)$ and $l \notin \Delta, l^{\perp} \notin \Delta$.

  Let π_1 be a partial proof-tree corresponding to $\Delta \| \phi$.

  We define π_2 as the parallel extension of π_1 according to an action $(\pi_a)_{a \in A}$ over the family of sequents labelling the open leaves of π_1, defined as follows:

  - for a sequent s_a labelling an open leaf mapped to $\Delta_0 \in [\![\Delta]\!]$, we define π_a as a 1-node tree labelled with s_a
  - for a sequent $s_a = \quad \Delta', \phi', \Gamma \vdash^{\mathcal{P}}$ labelling an open leaf mapped to $|\Delta|$, then:
    - If neither $l \in \mathcal{P}$ nor $l^{\perp} \in \mathcal{P}$, we define π_a as
      $$\frac{\Delta', \phi', \Gamma \vdash^{\mathcal{P}, l}}{\Delta', \phi', \Gamma \vdash^{\mathcal{P}}}$$
    - If $l \in \mathcal{P}$ then we define π_a as a 1-node tree labelled with s_a, and the open leaf is now mapped to $|\Delta, l|$.
    - If $l^{\perp} \in \mathcal{P}$ then we close the branch altogether by defining π_a as:
      $$\overline{\Delta', \phi', \Gamma \vdash^{\mathcal{P}}}$$
      as indeed $l^{\perp} \in \mathcal{P} \subseteq \mathsf{Sat}_{\phi}(\Delta')$ and $l \in \mathsf{Sat}_{\phi}(\Delta) \subseteq \mathsf{Sat}_{\phi}(\Delta')$ entails $\Delta' \models_{\mathcal{T}}$.

  We argue that π_2 is a 2-extension of π_1 that corresponds to $\Delta, l \| \phi$.